# Breast Cancer Classifier from Image Analysis of Cell Nuclei Data using XGBoost Model

The following notebook demonstrates a complete machine learning workflow using the Breast Cancer Wisconsin dataset. The steps include data exploration, preprocessing, training an XGBoost classifier, and evaluating the model's performance.

Objective:

- Fetch/inspect dataset
- No missing values in this dataset, encode categorical target variable to numerical
- Train an XGBoost model
- Evaluate model performance and feature importance:
  - Feature importance plot
  - Masked correlation matrix
  - Confusion Matrix
- Explore correlation matrix for any continued investigation or dimension reduction
- Discussion, Conclusions, Potential Next Steps

In [10]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
```

# Fetch and Inspect the Data

Download dataset from the UCI Machine Learning Repository via https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic become familiar with the structure and inspect data for missing values.

In [2]:
```
# fetch dataset
data = pd.read_csv('C:\\Users\\alexm\\Desktop\\Data\\breast_cancer_data.csv')

# Display the first 5 rows
print("Display first 5 rows: \n",data.head(5))
```

```
# Display summary stats
print("Display Summary Statistics: \n", data.describe())

# Display basic info
print("===>Display basic information about the dataset:\n", data.info())

#Check for missing values
print("missing values:\n",data.isnull().sum())
```

```
Display first 5 rows:
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0   
1    842517         M        20.57         17.77          132.90     1326.0   
2  84300903         M        19.69         21.25          130.00     1203.0   
3  84348301         M        11.42         20.38           77.58      386.1   
4  84358402         M        20.29         14.34          135.10     1297.0   

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710   
1          0.08474           0.07864          0.0869              0.07017   
2          0.10960           0.15990          0.1974              0.12790   
3          0.14250           0.28390          0.2414              0.10520   
4          0.10030           0.13280          0.1980              0.10430   

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622   
1  ...          23.41           158.80      1956.0            0.1238   
2  ...          25.53           152.50      1709.0            0.1444   
3  ...          26.50            98.87       567.7            0.2098   
4  ...          16.67           152.20      1575.0            0.1374   

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601   
1             0.1866           0.2416                0.1860          0.2750   
2             0.4245           0.4504                0.2430          0.3613   
3             0.8663           0.6869                0.2575          0.6638   
4             0.2050           0.4000                0.1625          0.2364   

   fractal_dimension_worst  Unnamed: 32  
0                  0.11890          NaN  
1                  0.08902          NaN  
2                  0.08758          NaN  
3                  0.17300          NaN  
4                  0.07678          NaN  

[5 rows x 33 columns]
Display Summary Statistics:
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000   569.000000   
mean   3.037183e+07    14.127292     19.289649       91.969033   654.889104   
std    1.250206e+08     3.524049      4.301036       24.298981   351.914129   
min    8.670000e+03     6.981000      9.710000       43.790000   143.500000   
25%    8.692180e+05    11.700000     16.170000       75.170000   420.300000   
50%    9.060240e+05    13.370000     18.840000       86.240000   551.100000   
75%    8.813129e+06    15.780000     21.800000      104.100000   782.700000   
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000   

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000   
mean          0.096360          0.104341        0.088799             0.048919   
std           0.014064          0.052813        0.079720             0.038803   
min           0.052630          0.019380        0.000000             0.000000   
25%           0.086370          0.064920        0.029560             0.020310   
50%           0.095870          0.092630        0.061540             0.033500   
75%           0.105300          0.130400        0.130700             0.074000   
```

```
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean  ...  texture_worst  perimeter_worst   area_worst  \
count     569.000000  ...     569.000000       569.000000   569.000000   
mean        0.181162  ...      25.677223       107.261213   880.583128   
std         0.027414  ...       6.146258        33.602542   569.356993   
min         0.106000  ...      12.020000        50.410000   185.200000   
25%         0.161900  ...      21.080000        84.110000   515.300000   
50%         0.179200  ...      25.410000        97.660000   686.500000   
75%         0.195700  ...      29.720000       125.400000  1084.000000   
max         0.304000  ...      49.540000       251.200000  4254.000000   

       smoothness_worst  compactness_worst  concavity_worst  \
count        569.000000         569.000000       569.000000   
mean           0.132369           0.254265         0.272188   
std            0.022832           0.157336         0.208624   
min            0.071170           0.027290         0.000000   
25%            0.116600           0.147200         0.114500   
50%            0.131300           0.211900         0.226700   
75%            0.146000           0.339100         0.382900   
max            0.222600           1.058000         1.252000   

       concave points_worst  symmetry_worst  fractal_dimension_worst  \
count            569.000000      569.000000               569.000000   
mean               0.114606        0.290076                 0.083946   
std                0.065732        0.061867                 0.018061   
min                0.000000        0.156500                 0.055040   
25%                0.064930        0.250400                 0.071460   
50%                0.099930        0.282200                 0.080040   
75%                0.161400        0.317900                 0.092080   
max                0.291000        0.663800                 0.207500   

       Unnamed: 32  
count          0.0  
mean           NaN  
std            NaN  
min            NaN  
25%            NaN  
50%            NaN  
75%            NaN  
max            NaN  

[8 rows x 32 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
```

```
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
===>Display basic information about the dataset:
 None
missing values:
 id                           0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
```

```
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

# Data Preprocessing

1. Drop uneeded column for a more efficient model
2. Create Feature and Target variables
3. Convert categorical target variable (M = malignant, B = benign) to binary

In [3]:
```python
# Drop the ID column
data = data.drop(['Unnamed: 32', 'id'], axis=1)

# Create Features and Target Variable
X =data.drop('diagnosis', axis=1)
y = data['diagnosis']

#convert Target variable to binary
y = y.replace({'M':1, 'B':0})
```

# Data Splitting, Scaling, and XGboost Training

Splitting the data is necessary for training and evaluating the preformance of the model. Normalizing the features with StandardScaler() can enhance the performance and convergence speed of XGboost algorithm.

stratify=y in the train_test_split function is important for maintaining the same proportion of classes in both the training and testing sets as in the original dataset. Here's a detailed explanation:

In [16]:
```python
# split the dataset for training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y

# Standardize the variables
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)

#Initialize the XGBoost Classifier
model = XGBClassifier()

# Train the model
```

```
model.fit(X_train, y_train)

# make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.98
Classification Report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98       107
           1       1.00      0.94      0.97        64

    accuracy                           0.98       171
   macro avg       0.98      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171
```

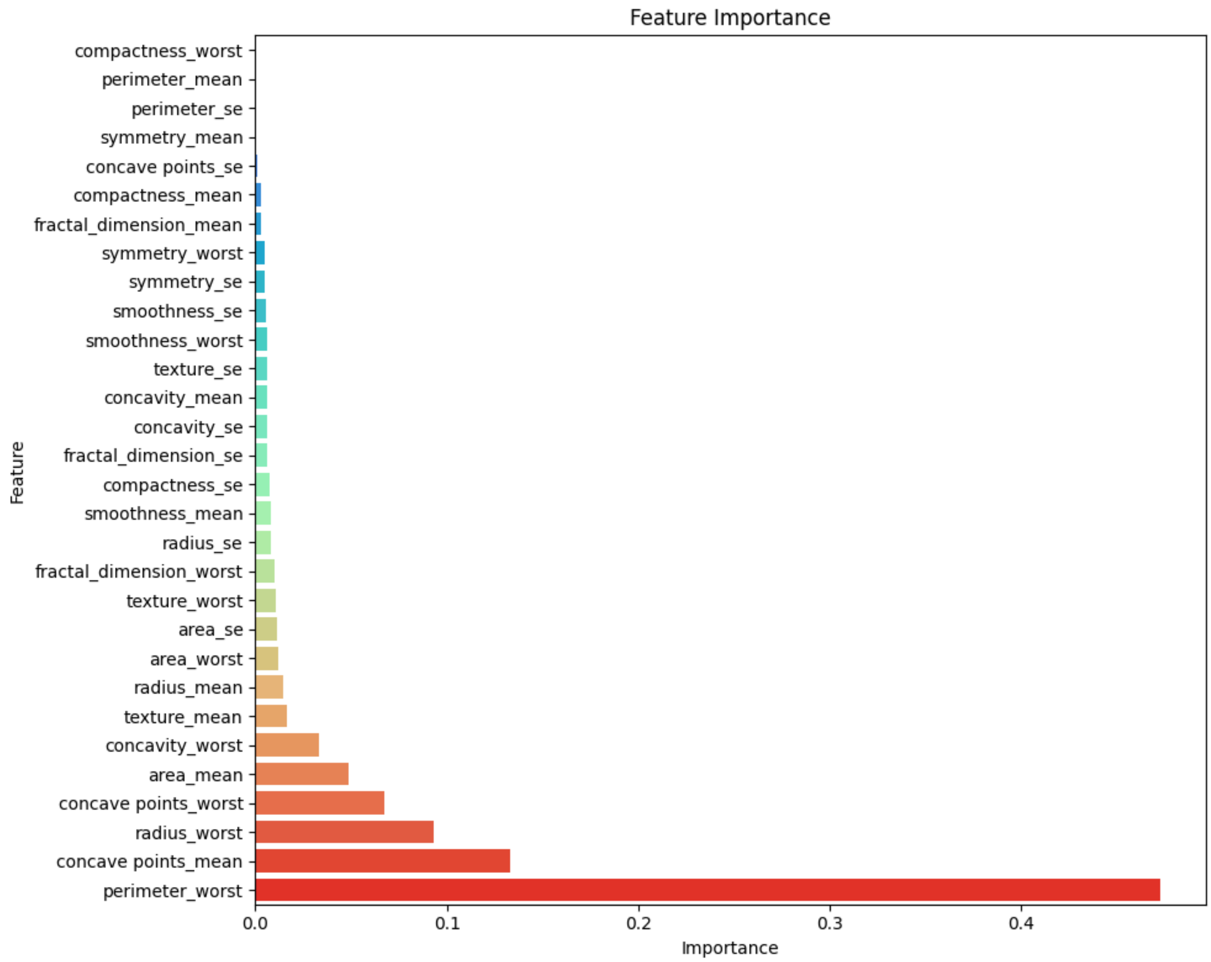
# Data Visualization

Visualization enables us to explore the data in a more digestible way and evalute the performace in a more intuitively.

1. Plot Feature Importance

Understand what features contribute most to the models performance.

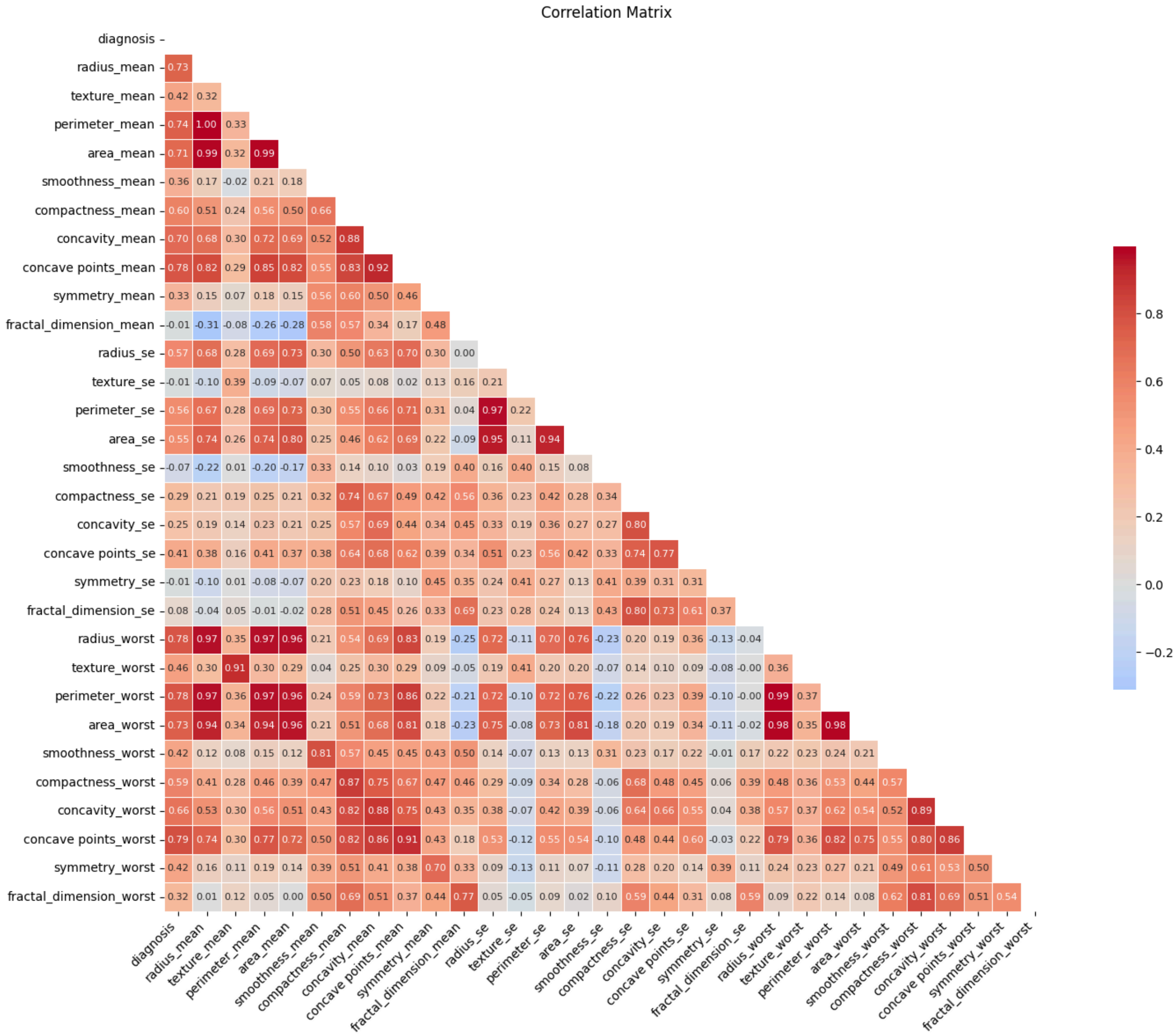
In [17]:
```
# Get feature importance
importance = model.feature_importances_
# Convert into a pandas DataFrame
features = pd.DataFrame({'Feature': X.columns, 'Importance': importance})
# Sort the features by importance
features = features.sort_values(by='Importance', ascending=True)
# Plot feature importance
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=features, palette='rainbow')
plt.title('Feature Importance')
plt.tight_layout()
plt.show()
```




2. Correlation Matrix Observe the realtionship of the feature vs. the target. Understand redunancy can be cleaned up for further model efficiency. A mask was applied for more intuitive viewing by eliminating redundancy.

In [18]:
```
# Convert Target categorical variable to numerical binary
data['diagnosis'] = data['diagnosis'].replace({'M': 1, 'B': 0})

# Calculate the correlation matrix
corr = data.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(16, 12))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap='coolwarm', center=0, annot=True, fmt=".2f",
            square=True, linewidths=0.5, cbar_kws={"shrink": 0.5}, annot_kws={"size

plt.title('Correlation Matrix')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)
```

```
plt.tight_layout()
plt.show()
```

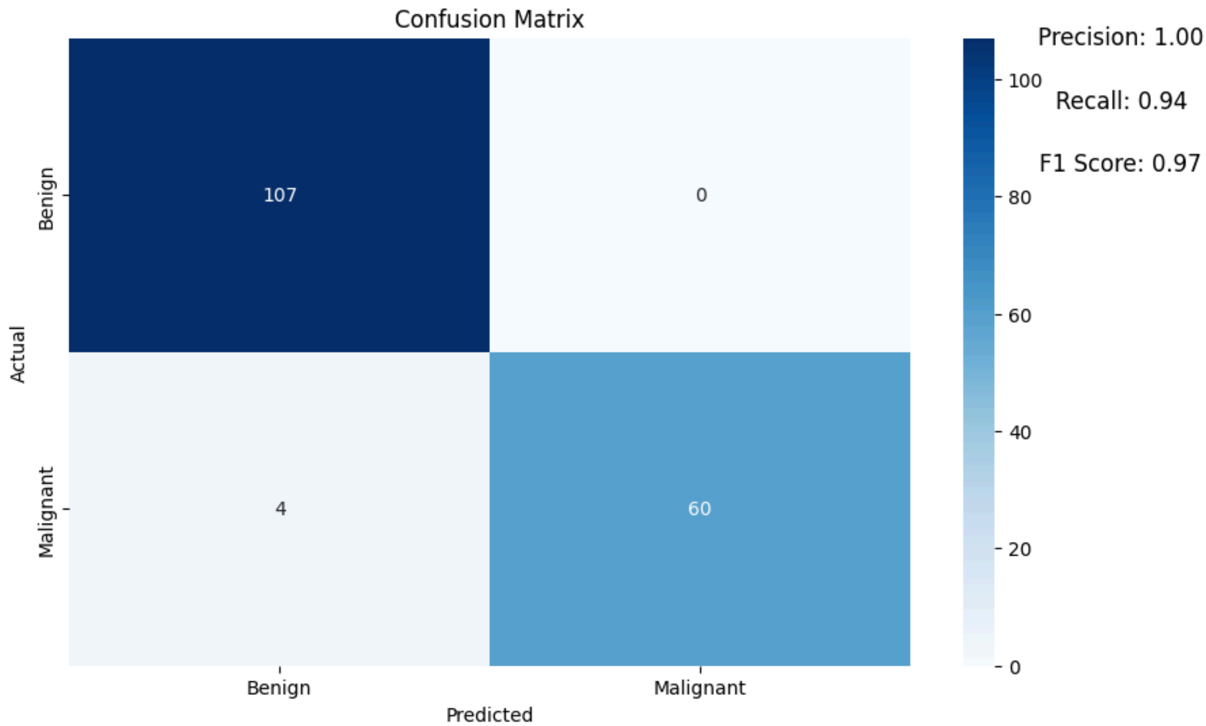



3. Confusion Matrix

Shows the number of true positives, true negatives, false positives, and false negatives.

In [20]:
```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Compute precision, recall, and F1 score
precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred, average=

# Plot confusion matrix with additional metrics
plt.figure(figsize=(10, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Benign', 'Malignant'],
            yticklabels=['Benign', 'Malignant'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
# Adding the precision, recall, and F1 score text to the plot
```

```
plt.text(2.5, 0, f'Precision: {precision:.2f}', ha='center', va='center', fontsize=
plt.text(2.5, .2, f'Recall: {recall:.2f}', ha='center', va='center', fontsize=12)
plt.text(2.5, .4, f'F1 Score: {f1:.2f}', ha='center', va='center', fontsize=12)
plt.show()
```




```
1.0 0.9375 0.967741935483871
```

# Conclusions

1. Model Performance: The XGBoost model used in the analysis demonstrated a high accuracy score of 98%, high precision (1.0) and very good recall (0.9375), resulting in an F1 score of approximately 0.9677. This indicates that the model is highly effective at identifying malignant cases without producing many false positives. High accuracy in the model suggests that it is reliable for predicting breast cancer diagnosis, with a low probability of misclassification.

2. Feature Importance: The feature importance plot highlighted perimeter_worst,concave points_mean, radius_worst were most influential in predicting the diagnosis. This information can be used to focus on the most critical factors in future studies or to reduce model complexity with dimensional reduction. Furthermore, if we were to look upstream to optimize the image recognition software to target the predictor feature more intensely this could benefit the workflow accuracy.

3. Correlation Analysis: The correlation matrix showed how features are related to one another. Highly correlated features can be redundant and may lead to inefficiencies in the model, tree-based models like XGBoost are generally robust to these issues.

# Discussion

1. Ethical Considerations: Using machine learning models for medical diagnoses requires careful consideration of ethical implications. False negatives, where the model fails to identify a malignant case ( 2% observed herer), could lead to serious consequences. Therefore, the model must be thoroughly tested and of course augmented with human oversight.

2. Generalization: The model's generalization to different populations or data sources needs to be carefully evaluated. Differences in demographics, medical practices, or data collection methods can affect the model's performance.

3. Collaboration with Medical Experts: Collaboration with healthcare professionals is crucial to ensure the model's predictions align with clinical insights and that the model is integrated into the diagnostic workflow appropriately.

# Next Steps

1. Model Optimization:
    - Hyperparameter Tuning: Hyperparameter tuning was not explored yet for this dataset. The XGboost model could be optimized.
2. Feature Engineering:
    - Feature Selection: Perform additional feature selection or dimensionality reduction techniques (e.g., PCA) to simplify the model and possibly improve its performance should be considered if the model was to be scaled.
    - Interaction Features: Investigate creating interaction features, which might capture more complex relationships between features and the target variable or highlight areas for more study/optimization.
3. Deployment:
    - Real-world Testing: If there is desire for the model deployed in a clinical setting, it should be tested with more real-world data to ensure it performs as expected outside of the controlled environment set that was observed here.
    - Model Monitoring: Set up a monitoring system to track the model's performance over time, ensuring that it continues to perform well as new data becomes available.